Date=15/12/2020
Lecture By= Akash Handa
Subject ⇒ React forms

IN PREVIOUS LECTURE **(QUICK RECAP) Date-14/12/2020**	**In Today's Lecture (Overview)**
What is Filter in React? Anxious In react React Slick Quick start	Forms in React Controlled Components Handling Multiple Inputs JSON Server Getting started

Forms in React

HTML form elements work a little bit differently from other DOM elements in React, because form elements naturally keep some internal state. For example, this form in plain HTML accepts a single name:

```
</body>
</html>
<form>
    <label>
```

```
      Name:
      <input type="text" name="name" />
    </label>
    <input type="submit" value="Submit" />
  </form>
```

This form has the default HTML form behavior of browsing to a new page when the user submits the form. If you want this behavior in React, it just works. But in most cases, it's convenient to have a JavaScript function that handles the submission of the form and has access to the data that the user entered into the form. The standard way to achieve this is with a technique called "controlled components".

Controlled Components

In HTML, form elements such as `<input>`, `<textarea>`, and `<select>` typically maintain their own state and update it based on user input. In React, mutable state is typically kept in the state property of components, and only updated with `setState()`.

We can combine the two by making the React state be the "single source of truth". Then the React component that renders a form also controls what happens in that form on subsequent user input. An input form element whose value is controlled by React in this way is called a "controlled component".

For example, if we want to make the previous example log the name when it is submitted, we can write the form as a controlled component:

```
  class NameForm extends React.Component {
      constructor(props) {
        super(props);
        this.state = {value: ''};
```

```
        this.handleChange = this.handleChange.bind(this);
        this.handleSubmit = this.handleSubmit.bind(this);
      }

      handleChange(event) {
        this.setState({value: event.target.value});
      }

      handleSubmit(event) {
        alert('A name was submitted: ' + this.state.value);
        event.preventDefault();
      }

      render() {
        return (
          <form onSubmit={this.handleSubmit}>
            <label>
              Name:
              <input type="text" value={this.state.value}
onChange={this.handleChange} />
            </label>
            <input type="submit" value="Submit" />
          </form>
        );
      }
    }
```

Handling Multiple Inputs

When you need to handle multiple controlled `input` elements, you can add a `name` attribute to each element and let the handler function choose what to do based on the value of `event.target.name`.

For example:

```
  class Reservation extends React.Component {
       constructor(props) {
         super(props);
         this.state = {
           isGoing: true,
           numberOfGuests: 2
         };

         this.handleInputChange = this.handleInputChange.bind(this);
       }

       handleInputChange(event) {
         const target = event.target;
         const value = target.type === 'checkbox' ? target.checked :
target.value;
         const name = target.name;

         this.setState({
           [name]: value
         });
       }

       render() {
         return (
           <form>
             <label>
               Is going:
               <input
                 name="isGoing"
                 type="checkbox"
                 checked={this.state.isGoing}
                 onChange={this.handleInputChange} />
             </label>
             <br />
             <label>
               Number of guests:
               <input
                 name="numberOfGuests"
                 type="number"
                 value={this.state.numberOfGuests}
```

```
                onChange={this.handleInputChange} />
            </label>
          </form>
        );
      }
    }
```

JSON Server

Get a full fake REST API with zero coding in less than 30 seconds (seriously)

Created with <3 for front-end developers who need a quick back-end for prototyping and mocking.

Getting started

Install JSON Server

```
npm install -g json-server
```

Create a `db.json` file with some data

```
{

  "posts": [

    { "id": 1, "title": "json-server", "author": "typicode" }

  ],

  "comments": [
```

```
    { "id": 1, "body": "some comment", "postId": 1 }


  ],


  "profile": { "name": "typicode" }


}
```

Start JSON Server

```
json-server --watch db.json
```

Now if you go to http://localhost:3000/posts/1, you'll get

```
{ "id": 1, "title": "json-server", "author": "typicode" }
```